Mathematical Tools for Neuroscience (NEU 314)
Princeton University, Spring 2016
Jonathan Pillow

Homework 8: Logistic Regression & Information Theory

Due: Tuesday, April 26, 9:59am

Optimization Toolbox

One of the things to learn in this problem set is how to use the optimization toolbox. Specifically, we'll use the function `fminunc`, which performs "unconstrained minimization of a function". It allows you to pass in a function and a starting point (a vector), and it performs gradient descent of the function to find the location of a local minimum.

The optimization tooldbox functions generally do minimization instead of maximization. Since we generally want to maximize a likelihood function, you'll actually end up writing a function to compute the *negative* log-likelihood (since the maximum of the log-likelihood is in the same place as the minimum of the negative log-likelihood).

Before diving into the problem set, it will help to familiarize yourself with how to use `fminunc`. Open up the script `example_OptimizationScript.m`, which shows the basic steps involved in using `fminunc` to minimize an example function I wrote (which appens to be a quadratic) called `exampleLossFunction.m`. Open up this function too, just to see what it computes (and what good commenting practice looks like when you write your own functions!).

You should be able to use `example_OptimizationScript.m` as a starting point for coding up maximum likelihood estimation of a logistic regression model.

Logistic regression

The logistic regression is an extremely popular and important model for binary classification. This comes up when we have a stack of vectors with labels "0" and "1" and we'd like to find a linear hyperplane that optimally separates them (or equivalently, a 1D linear projection that captures the probability of any vector being a 0 or 1).

Assume we have data vectors and labels $\{(\mathbf{x}_i, y_i)\}$ for $i = 1, \ldots, N$, where $\mathbf{x}_i \in \mathbb{R}^d$ is a d-dimensional input vector, and $y_i \in \{0, 1\}$ is the corresponding label ("0" or "1") for that vector. For our purposes, think of $\mathbf{x}$ as a stimulus for a single time bin (e.g., an image represented as a length-d vector) , and y as indicating whether a neuron spiked or not.

The logistic regression model has a parameter vector $\mathbf{w}$, which we can think of as the neuron's receptive field (i.e., a linear set of weights telling us how the neuron integrates each element of the

stimulus) and a constant offset b, which will tell us the neuron's bias (towards firing or silence) in response to a zero-vector input.

To compute the probability of a spike under the logistic regression model, we take a dot product between $\mathbf{x}$ and the weight vector $\mathbf{w}$, and add b. Then we put this value through a logistic function:

$$P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{x}^\top \mathbf{w} - b\right)}. \tag{1}$$

The probability of spike plus the probability of no-spike has to sum to 1, so we have

$$P(y = 0|\mathbf{x}) = 1 - P(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(\mathbf{x}^\top \mathbf{w} + b)}. \tag{2}$$

Verify for yourself that these numbers add to 1.

We can express these probabilities more compactly as:

$$P(y|\mathbf{x}) = \frac{\exp(y(\mathbf{x}^\top \mathbf{w} + b))}{1 + \exp(\mathbf{x}^\top \mathbf{w} + b)} \tag{3}$$

Verify for yourself that this agrees with the probabilities given in equations 1 and 2 above.

Your goal for the first set of problems will to be to write a function that evaluates the negative log-likelihood of a dataset. Taking the negative of the log of (eq. 3) and summing across all stimulus-response pairs, we can show this to be equal to:

$$-\log P(Y|X, \{\mathbf{w}, b\}) = -\sum_{i=1}^{N} y_i(\mathbf{x}_i^\top \mathbf{w} + b) \quad + \sum_{i=1}^{N} \log(1 + \exp(\mathbf{x}_i^\top \mathbf{w} + b)) \tag{4}$$

You will hand this function to `fminunc` and ask it to find the values of $\mathbf{w}$ and b that minimize it (i.e., the maximum-likelihood estimates $\hat{\mathbf{w}}_{ML}$ and $\hat{b}_{ML}$.).

To do this, you will want to create a single vector parameter vector that contains both $\mathbf{w}$ and b inside it, e.g., `params = [w;b];`.

1. **Load and plot the raw data**

   Load the dataset `neuralencodingdataset1.mat`. This contains 1000 stimulus-response pairs from a (simulated) neural experiment. The matrix x is a 1000×2 matrix, each row of which is a 2-component stimulus vector for a single trial (or "time bin"). The vector y holds the corresponding spike count (or "label") observed for each stimulus.

   Now create a "training set" consisting of the first 800 datapoints, e.g., `xtrain = x(1:800,:); ytrain = y(1:800);` Set aside the final 200 points as a "test" dataset that you'll use later when evaluating the quality of your model fit.

   Make two scatter plots of the raw data, using different symbols for the "spike" and "no-spike" stimuli. One plot should show the training data, the other should show the test data.

2. **Warmup: least-squares regression estimate**

(a) Compute the least-squares estimate $\hat{\mathbf{w}}_{LS}$ for the weight vector $\mathbf{w}$ under the linear-Gaussian regression model of the data, given by: $y_i = \mathbf{x}_i^\top \mathbf{w} + n_i$, where n_i is an independent Gaussian random variable with mean zero and variance σ^2. Use the training data for this. (You should be able to do this with a single line of matlab. The formula should be second-nature to you by now!)

(Note: we know that a linear-Gaussian model is the wrong model for this data, since Gaussian noise will not produce outputs with the values 0 and 1. But still, it's (perhaps) not a bad thing to start with.)

(b) Re-generate the scatter plot of the training data and add a unit vector pointing in the direction of the estimated $\hat{\mathbf{w}}_{LS}$ vector. (That is, compute a unit vector pointing in the same direction as $\hat{\mathbf{w}}_{LS}$ and add a line from 0 to the corresponding point on the unit circle). Does it look like it points in the correct direction, i.e., the direction that will preserve information about the probability of spiking if we project the stimuli onto it?

3. **Logistic regression: maximum likelihood estimation**

(a) Write a function to evaluate the negative log-likelihood of the dataset under a logistic regression model given parameters $\mathbf{w}$ and b, as described above. It would be great if you can avoid `for` loops! (2 points).

(b) Use `fminunc` to find the maximum likelihood estimate of $\mathbf{w}$ and b under the logistic model given the training data. (See notes above on optimization toolbox and logistic regression model).

(c) Re-generate your scatter plot of the training data and add a unit vector pointing in the direction of $\hat{\mathbf{w}}_{ML}$.

(d) Now add a line showing the linear separatrix, that is the line dividing the region of x space where $P(y = 1|\mathbf{x}) < 0.5$ from the region where $P(y = 1|\mathbf{x}) > 0.5$. (You will need to do a little bit of math to figure out what this line is; Your two hints are that it corresponds to the points for which $P(y = 1|\mathbf{x}) = 0.5$, and it lies orthogonal to $\hat{\mathbf{w}}_{ML}$.) (2 points).

(e) Make a separate plot showing the projected training datapoints $\mathbf{x}^\top \mathbf{w} + b$ on the x axis, and the probability $P(y = 1|\mathbf{x})$ under the fitted model on the y axis. Use a different symbol/color for the points with $y = 0$ from those with $y = 1$. Now add a smooth plot of the fitted logistic function using a grid of x values extending over the range of the data.

(f) Classify both training and test datasets by assigning labels to each point. Assign "z =0" if $P(y = 1|\mathbf{x}) < 0.5$ and "z =1" otherwise. Now compute the classification error the training data and the test data by comparing the "true" labels y with the model-assigned labels z. Did your model do better on the training or test data?

(g) Compute the (positive) log-likelihood per sample for both the training and test sets. That

is, evaluate the negative log-likelihood of the training sets uing parameters $\{\hat{\mathbf{w}}_{ML}, \hat{b}\}$, then divide by 800, the number of training points and multiply by (-1). Do the same for the test set (using the same parameters), but divide by the number of test points. These quantities are known as the "training log-likelihood" and "test log-likelihood" per sample. Which is higher? Why do you think this is?

4. **Nonlinear classification using logistic regression.**

   Load the dataset `neuralencodingdataset2.mat`, which contains `x` and `y` of the same format as the previous problem.

   (a) Make a scatter plot of the raw data, using different symbols for points labelled 0 and 1. What do you notice about the separability of these data? (Could they be well separated with a line?)

   (b) Repeat steps b,c,d, and f from the previous problem on the new dataset (using the first 800 samples for training and the last 200 for testing, just as before). You should be able to copy the code from the previous problem over a new file and just call it with the new data.

   (c) Augment the `x` matrix with three additional columns, given by quadratic functions of the original input vectors: `x(:,1).^2`, `x(:,2).^2`, and `x(:,1).*x(:,2)`. These columns provide a basis set for all quadratic functions of the original x, and will allow us to fit an arbitary quadratic classification boundary between the points labelled $y = 0$ and $y = 1$. The new design matrix will have 5 columns. Run logistic regression on this expanded data, using the same division between training and test data. (That is, find the maximum likelihood estimators for $\mathbf{w}$ and b under the logistic regression model, where $\mathbf{w}$ is now vector with 5 components.)

   (d) Compute the training and test error of the "quadratic" logistic regression model. How does it compare to the training and test error of the original model?

   (e) Make a pair of scatter plots of the test data using distinct symbol (or color) for the 4 different kinds of points: points correctly labelled 0 ("correct rejects"), points incorrectly labelled 0 ("misses"), points correctly labelled 1 ("hits") and points incorrectly labelled 1 ("false alarms"). Make one plot using the labels from the original (linear) logistic regression model, and another plot using labels from the quadratic logistic regression model.

Information Theory

5. Load the file `NeuralResps.mat`, which contains data from a simulated neuroscience experiment. The vector S is a series of stimulus intensities presented to a neuron, and R is a vector of corresponding responses (which you can assume to have been simulated iid from $P(R|S)$).

(a) Compute the joint distribution $P(R, S)$ and marginal distributions $P(R)$ and $P(S)$. (You might wish to use the function `hist3`). Plot the two marginals, and use `imagesc` to display the joint.

Now use these sample distributions to compute plug-in estimates of:
(b) the entropy $H(S)$;
(c) the entropy $H(R)$;
(d) the mutual information $I(S, R)$.

6. Generate a 2D discrete probability table (of dimension 50×50) for the joint distribution $P(X, Y)$ of two random variables X and Y. Generate a joint distribution such that X and Y are completely uncorrelated but have high mutual information. Make a plot of $P(X, Y)$ and compute the correlation coefficient and the mutual information between X and Y.